SURVE Toolkit: the User's Guide

William Spears, Diana Spears, Wesley Kerr,
Suranga Hettiarachchi, Dimitri Zarzhitsky

Department of Computer Science
University of Wyoming, Laramie, WY 82071

November 19, 2006

1 Introduction

This manual covers the SURVE simulator for automated outdoor surveillance, focusing on beginner and intermediate concepts needed to install the SURVE toolbox, conduct experiments, and to collect data. The SURVE toolkit is intended to facilitate the study of a surveillance problem, where several mobile aerial *assets* attempt to detect the presence of ground-based *targets*, which may or may not be moving. To simplify the discussion, this manual assumes that the assets are uninhabited aerial vehicles (UAV), charged with detection of invading vehicles such as tanks and supply convoys, that may attempt to conceal their presence by hiding under the tree canopy or by avoiding the UAV sensors.

Thus the toolkit is a test bed for different types of surveillance problems, and is distributed with several realistic control strategies for both the assets and targets. Users can run interactive experiments, with real-time feedback via an advanced GUI, and can execute a pre-programmed set of experiments in a non-interactive batch mode. Simple simulation statistics are computed within the simulator, and raw simulation data can be stored for later post-processing and in-depth analysis.

This document first outlines how to install the SURVE toolkit and to run basic experiments in the interactive mode, and then explains how to create experiment sets for batch processing. It also provides a detailed overview of the GUI controls and the configuration files. Please keep in mind that SURVE software is still in development, and thus minor inconsistencies or unresolved issues may persist in the working version of the program.

2 Quick Start

The SURVE toolkit is written using the multi-platform Java2 programming language, and is distributed in the form of a ZIP and TAR archives. It should run on any platform for which a Java Runtime Environment (JRE) is available, and has been extensively tested with Linux and Windows operating environments. To install the software, simply uncompress the distribution file into an empty directory (consult your operating system manual or system administrator for assistance with this step). The distribution archive includes all of the compiled files along with the source code, and no further configuration is necessary.

To execute the SURVE simulator, change into the `surveillance2` directory that was created when you decompressed the distribution archive, and run the Java interpreter on the `graphics.simulator.Observer` class. If you're not sure how to execute Java programs on your computer, please consult the Java help page at http://java.sun.com/help/. If you use a UNIX-like operating environment, then simply execute the `runVis.sh` script from a command prompt. When you see a window similar to the one shown in Fig. 1

then you have successfully installed the SURVE toolbox, and can start running experiments, as outlined in the following sections.

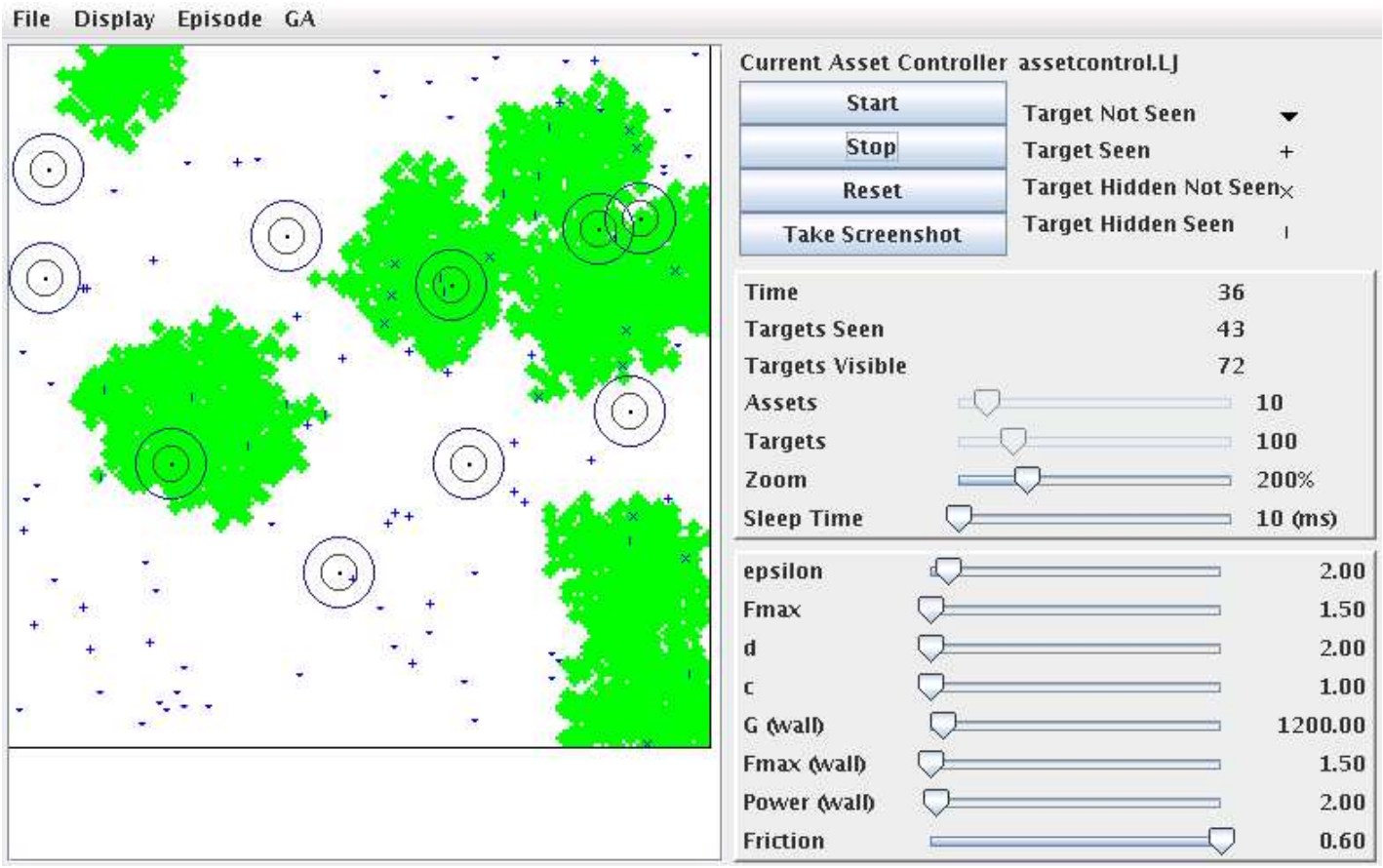
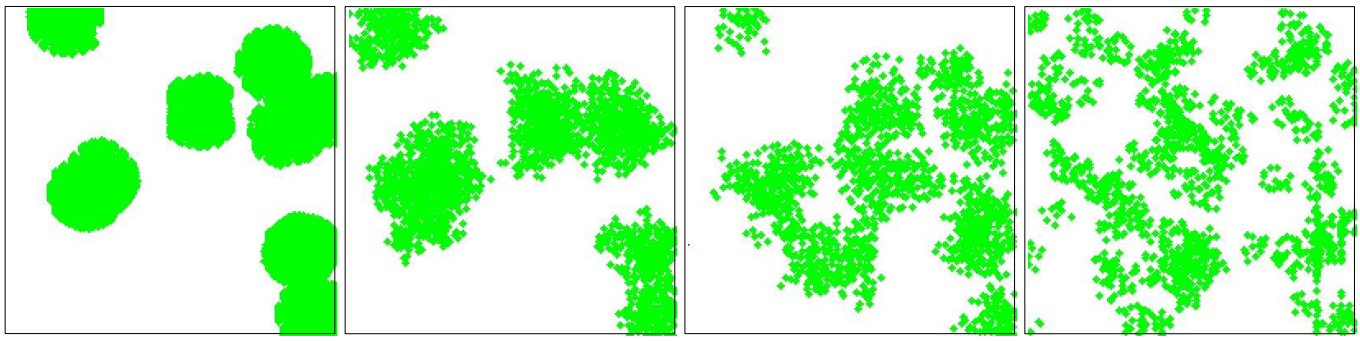


Figure 1: SURVE simulator window.

3 Surveillance Experiments

Before explaining how to use the SURVE toolkit to conduct surveillance experiments, it will be helpful to explain the methodology behind the design of the toolkit, so that it will be easy to understand the structure of the experiments. For the purposes of the toolkit, we assume that surveillance is taking place in a rectangular region. The landscape is assumed to be flat, but the presence of dense forest, impenetrable by the sensors, complicates the surveillance problem. Two basic surveillance scenarios are of interest: a case where the targets are stationary (perhaps awaiting deployment), and a case where targets move in some organized, directed fashion (as may be the case of a motorized assault or a resupply mission). In both scenarios, the monitoring UAVs move in a trajectory indicated by the pre-programmed surveillance strategy for a given length of time, consistent with the fact that UAVs have a time-of-flight limit imposed by limited energy resources.

During the surveillance activity, the UAVs continually scan the ground underneath them, and whenever the sensor beam detects a target, the target is marked as "seen" by the simulator as part of its data-collection activity. There are several user-specified parameters that affect how the target sensor operates, but for clarity, the discussion of these and other configuration parameters is presented in the following sections. Some of the asset strategies distributed with the SURVE toolkit also assume that the UAVs are equipped with forest sensors, in hopes that the prudent use of extra sensor information will increase the number of detected targets in a given time period.



seedRadius=5	seedRadius=7	seedRadius=8	seedRadius=10
seedStrength=0.5	seedStrength=0.007	seedStrength=0.02	seedStrength=0.95
seedDecay=0.5	seedDecay=0.7	seedDecay=0.98	seedDecay=0.95
forestSize=10	forestSize=7	forestSize=15	forestSize=90

Figure 2: Foliage distributions produced by the environment generator along with the values of the corresponding algorithm parameters. These images show a 30% total area coverage of the 200×200 world.

3.1 Building Simulated Environment

The first step in the surveillance scenario is the construction of the environment. Two main parameters characterize the area over which surveillance is conducted: the size, which is a single integer dimension defining a square-shaped area, and the extent of world foliage cover, specified as a floating point fraction less than one. World size is specified in the `conf/general.txt` configuration file, in the `Environment Parameters` section using the `size=`*integer* entry, where the positive integer value is the length of the world boundary in screen pixels. Keep in mind that the larger the world size, the greater the computational resources (memory and CPU power) that are needed to execute the simulator. The default world size is 200×200 units.

The foliage growth in the world creates areas where UAV target sensors are ineffective; the notion of simulated foliage may also be thought of as an abstraction of other environmental obstacles that may inhibit surveillance sensors (e.g., caves or tunnels within a mountain terrain), although the environment builder supplied with the current version of the surveillance simulator only mimics natural plant growth (e.g., a forest of trees). Environment generator has several parameters that control the type of foliage distribution produced, all of which are specified in the `conf/forest.txt` configuration file (see Fig. 2).

Different foliage maps can be created (and recreated) by specifying a nonnegative seed to the random number generator (look for `forestSeed=`*integer* in the `conf/forest.txt` file). The default value of -1 tells the simulator to use the current system time (in seconds) as the random seed, so that a different forest is generated each time the simulator is run, or when the forest is reset via the Episode menu (see Section 6 on simulator menus).

To control the foliage placement, first consider the following high-level description of the forest generation algorithm: the forest begins as a collection of randomly placed "trees", where the initial number of these starter trees is given by the `forestSize=`*integer* parameter in the `conf/forest.txt` file. Then, each tree produces "seeds" which fall to the ground surrounding a given tree within the area specified by the `seedRadius=`*integer* configuration parameter. Seed viability is controlled by the `seedStrength=`*float* value, which represents the probability that a given seed will produce a new tree. Note that a location with two seeds is twice as likely to produce a new tree than a location with just one seed, given equal seed strengths. As time passes, each seed ages and becomes less likely to grow into a tree; the rate at which the seeds weaken is specified by the `seedDecay=`*float* line in the forest configuration file. All grown trees continue to produce new seeds until the total world foliage cover is equal to the `forestCoverage=`*float* parameter. *Note:* be careful when using small values for `seedRadius` and `seedStrength` or large values

for `seedDecay` and `forestSize`, as it may be either impossible or may take a prohibitively long time to achieve the specified `forestCoverage` with seeds that are extremely weak, decay very quickly, or if the initial number of starter trees exceeds the world area; the ranges shown in Fig. 2 work well for many environments, and should be treated as a guide for future settings of these parameters.

3.2 Initializing Targets

After constructing the desired simulated environment, the next step in the experiment preparation process is the set up and initialization of targets – the objects which will be detected during the surveillance activity. Each target is assumed to have a 1×1 unit dimension, and may move with constant speed in any direction or remain stationary during the simulation. A target's movement trajectory is computed by a target controller, and currently, all targets in the simulation share the same target controller algorithm during any given run. However, a different target group behavior may be obtained quickly, without restarting the simulator, by simply switching the target controller via the File menu (see Section 6 on simulator menus).

In order to evaluate the performance of a given surveillance strategy, it is helpful to enumerate the relevant properties and states of the simulated targets. Each target, at any given time step, may be either visible or hidden. A visible target is located within the world boundaries at a point without foliage cover. Likewise, a hidden target is either outside of the world boundaries, or is located under the dense cover of the forest, and cannot be observed with the surveying equipment. For moving targets, the "hidden" and "visible" target states refer to the *current* state of a target, and may change many times during the simulation, unlike the "seen" and "unseen" classification, that may only change once. As the name implies, the "seen" category is used to describe targets that have been detected by a UAV target sensor, while "unseen" targets are the ones that have successfully escaped the detection. Note that all of simulated targets, including those that are stationary, start out in the unseen group, but will change their state to seen after being detected by the UAVs. Thus, one of the most important performance metrics used to assess the merit of any surveillance strategy during an experiment is the ratio of seen targets to all of the targets that have been visible (and thus potentially detectable) at least once during the run.

It will come as no surprise that the UAV surveillance performance depends on the movement strategy employed by the targets. In other words, a surveillance approach that does well in detecting stationary targets might do poorly in detecting moving targets, and vice versa. Therefore, the SURVE simulator is distributed with several realistic target controllers that can be selected at run time:

- *Avoid:* targets attempt to actively avoid the ground-level sensor footprint of the overhead UAVs while crossing the area under surveillance. When a moving target detects a UAV sensor beam in its path, it pauses for a short time, waiting for the UAV to move away; if the sensor beam persists after the delay, then the target will attempt to circumnavigate around the sensor footprint.

- *Ribbon:* targets follow a periodically sinuous trajectory, which varies slightly for each target. The amplitude and period of the mean-path oscillation are kept low to approximate realistic deviations from straight line travel due to terrain configuration.

- *Sneaky:* targets aim to minimize their exposure time to the UAV sensors by attempting to maximize the amount of travel time spent underneath the foliage. This target controller assumes that the targets may scan their local environment for foliage-covered areas, and will select the next waypoint to be inside the forest if possible.

- *Stationary:* targets simply do not move.

- *StraightLine:* targets move in a horizontal line formation from the left world boundary to the right edge of the surveillance area.

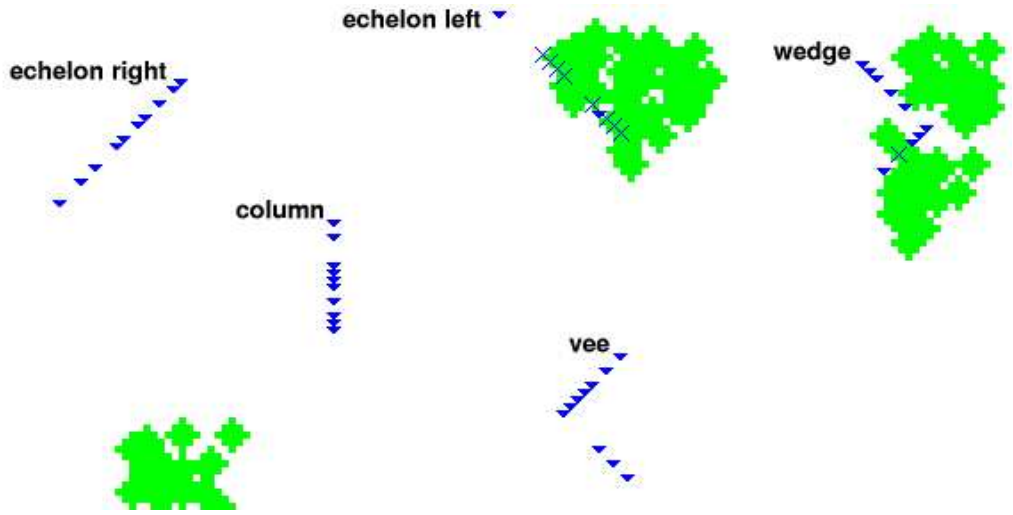


Figure 3: Targets arranged in the wedge, vee, echelon left/right, and column formations.

- *militaryCombo:* targets move in a combination of wedge, vee, echelon left/right, and column formations from the left world boundary to the right boundary of the surveillance perimeter. These target groupings mimic the actual tank battlefield formations, and a representative sample of each formation type is shown in Fig. 3.

Because of the varying nature of the trajectories generated by each target controller type, some configuration parameters differ significantly for each controller choice. However, one common aspect of target initialization is that it is done at random, using either a uniform or a normal probability distribution. The random number generator seed used by the target controllers is given by the `targetSeed=`*integer* parameter in the `conf/general.txt` configuration file; the default is a negative value, which yields a different random sequence each time the simulator is invoked). The number of targets in the simulation is controlled by the `targets=`*integer* parameter, with the default value of a 100. Target speed is specified by the `targetSpeed=`*float* parameter, and the units match world dimensions, so that a `StraightLine` target with a speed of 1.0 will cross a 200×200 world in 200 simulation steps, unless collision avoidance procedures slow down the moving target. The `targetSpeed` parameter is ignored by the `Stationary` targets.

Additional control of the targets is possible through the `targetOffset=`*float* parameter, which specifies the horizontal x coordinate location of the initial target placement random distribution center point in terms of the world size. For instance, a `targetOffset=0.0` setting places the center of the target distribution in the middle of the surveillance area, so that all targets start out within the surveillance area boundary (this should be the setting used for most experiments with the `Stationary` target controller). Setting `targetOffset` to -1.0 causes all of the targets to be initially placed before the left world edge, outside of the surveillance area, but still using the same random distribution (think of the target placement as just being shifted to the left by one world size); this is useful if the UAV units require time to organize, and a delay between the UAV deployment and the first target appearance is desired. Note that due to the fact that the standard SURVE controllers move targets from left to right, positive `targetOffset` values have little utility. The initial target controller may be selected via the `targetControl=`*string* parameter, and its value should be a fully-qualified Java class name, such as `targetcontrol.Stationary`; alternatively, the target controller may be changed via the File menu (see Section 6 on simulator menus).

3.3 Initializing Assets

Intelligent, efficient, and effective control of the surveillance UAVs is the major topic of interest driving this research. The chief aim is to devise a surveillance strategy that maximizes the target detection rate

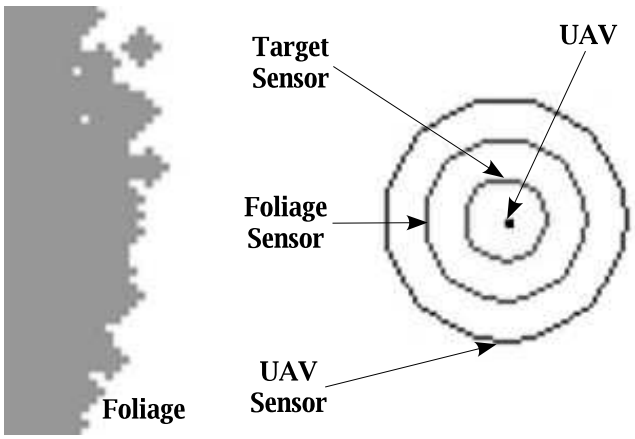


Figure 4: The model of a surveillance asset, showing the three on-board sensor types with their ranges.

by maximizing the benefits of asset-asset collaboration, sensor data utilization, and clever exploitation of the existing domain knowledge. Thus it is of little surprise that the thrust of the SURVE development work focused on the asset behavior.

A conceptual schematic of the SURVE UAV asset model is given in Fig. 4, which depicts a single asset in the vicinity of some foliage, and shows the ranges of the three types of sensors assumed to be present on every UAV in the simulation. The most basic aspect of the UAV model is the asset representation, which assumes an arbitrarily small asset size (a point model) with a real-valued coordinate location. The most important UAV sensor is of course the target sensor, whose radius is specified in the `conf/asset.txt` file by the `targetRadius=`*integer* parameter.

Because of the critical role the target sensor plays in the surveillance problem additional configuration options are available for customizing target sensor characteristics. The shape of the ground-level target sensor footprint is controlled by the `targetSensorType=`*string* parameter, with possible values of *circle* or *square*. In case of a *square* target sensor, the `targetRadius` parameter defines half the width of the ground-level sensor footprint, so that a setting of `targetSensorType=square` and `targetRadius=5` will produce a sensor with a 10×10 target detection area. The Boolean `targetSensorIncludeBounds=`*0|1* parameter is a fine-tuning option which controls whether the sensor will detect targets that are located right on the sensor footprint boundary. A value of 0 indicates that a strict $<$ (less than) comparison is to be used when computing distances, thus excluding targets that only touch the sensor boundary, and a value of 1 specifies that a $\leq$ (less than or equal) test be used throughout the simulation, allowing detection of such targets. Perhaps the most crucial parameter is the `detectionProb=`*float* specifier, which determines the percentage of sensor readings that correctly detect a target when it is in the view-field of the sensor. Thus a setting of 0.0 will inhibit all target detection (extreme sensor noise), and a setting of 1.0 will construct a perfect target sensor, which always properly identifies a target when the UAV passes over it.

Two other types of sensors shown in Fig. 4 are the asset-to-asset sensor and the foliage detector. These sensors are "optional" in the sense that only some of the built-in UAV controllers make use of them. In particular, interactions between assets are managed by controllers based on the *Artificial Physics* framework (visit www.cs.uwyo.edu/∼wspears/ap.html for more information). Likewise, information about foliage cover in the environment is exploited by only a few of the asset controllers included in the default SURVE distribution (the size of the circular area covered by the foliage sensor of each UAV is set by the `foliageRadius=`*integer* parameter in the `conf/asset.txt` configuration file). The list below provides the detailed information about each of the standard asset controllers:

- *AP:* is a physics-based controller which uses a generalized form of Newton's Law of Universal Gravitation, $\vec{F} = Gm_1m_2/r^p$, to compute a virtual force between asset 1 and asset 2, with respective

virtual masses of m_1 and m_2, separated by a distance r (see Fig. 5). In many cases, the assets are assumed to have the same virtual mass, along with some desired separation dictated by practical constraints, such that the pair-wise inter-asset force is attractive if the actual separation between UAVs exceeds the desired range, and becomes repulsive if the assets move too close to each other. By varying the parameters which define this force, a variety of dynamically stable UAV formations can be self-organized and maintained in a completely distributed fashion. The force law may be altered manually using the controls in the bottom-right window panel (see Section 6 on simulator controls), or alternatively, the appropriate set of values may be constructed using a Genetic Algorithm, and loaded into the simulator from the evolution history file via the GA menu (see Section 6 on simulator menus).

- *Bouncy:* is a controller motivated by the need to carry out statistical experiments, and simply moves the assets vertically, "bouncing" off the top and bottom world boundaries by reversing the UAV velocity upon reaching the edge of the surveillance area.

- *Column:* is a variation on plain vertical movement of the UAVs, except the world is assumed to be toroidal, so that the assets "wrap around" in the same vertical column when moving past the top and bottom edges of the surveillance area. Modeling such behavior is helpful when performing statistical analysis of the surveillance problem.

- *LJ:* is another physics-based asset controller which uses a generalized Lennard-Jones force function:

$$\vec{F} = 24\varepsilon \left[\frac{d(\sigma_1 + \sigma_2)^6}{r^7} - \frac{2c(\sigma_1 + \sigma_2)^{12}}{r^{13}} \right]$$

  to facilitate UAV formation building (see Fig. 6). The two fractions in the above formula control the balance between the attractive and repulsive components of the force with respect to the current

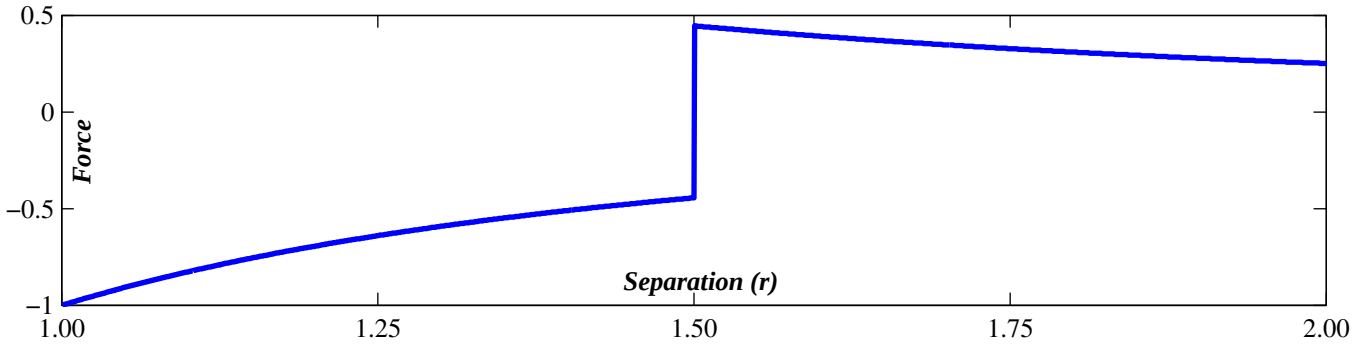


Figure 5: Plot of a sample force function used by the AP asset controller, with $r_{desired} = 1.5$ and $p = 2$.

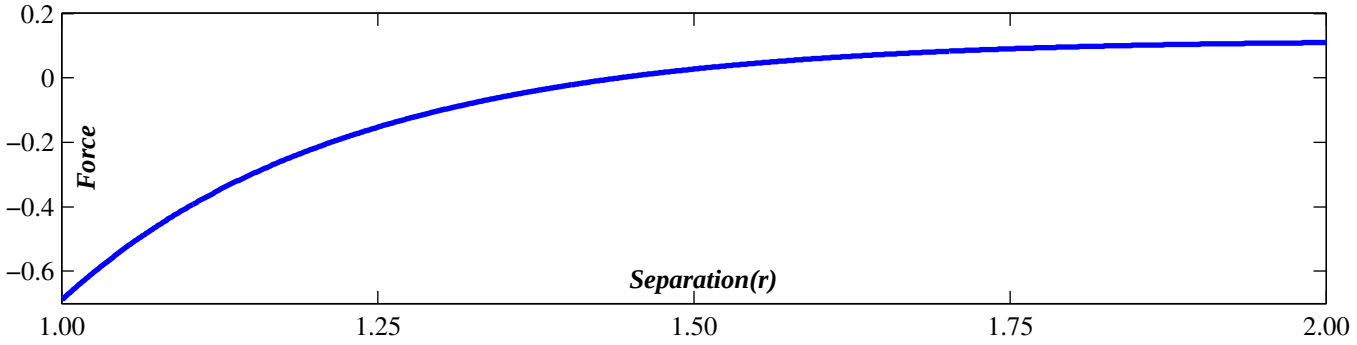


Figure 6: Plot of a sample force function used by the LJ asset controller, with $\sigma_1 = \sigma_2 = 1.5$.

UAV separation radius r and the desired separation radii σ_1 and σ_2 for each pair of assets. As in the case of the *AP* controller, parameters may be set with the slider bars on the front panel, or can be loaded from a GA evolution trace file. Note that this controller uses the AP force law when computing the force between an asset and a surveillance area boundary.

- *LJMass:* is very similar to the *LJ* asset controller, but it attempts to account for the foliage growth (using the foliage sensor mounted on each UAV) by slowing down the UAV near the boundaries of the forest.

- *MagicCarpetArea:* is an asset controller useful in testing statistical models of the surveillance problem, in which assets "appear" at random locations throughout the world during the simulation. UAV locations are selected using a uniform random distribution such that target sensor footprints do not overlap. This controller ignores the maximum asset speed specified in the `conf/asset.txt` file.

- *MagicCarpetColumn:* similar in spirit to the *MagicCarpetArea*, this controller causes assets to appear at random locations, but restricts the surveillance region to a single column, so that the x coordinate of the asset remains fixed, as the y coordinate changes randomly, without target sensor footprint overlap between adjacent asset locations.

- *SL:* a realistic asset controller which ignores foliage distribution, and simply moves an asset in a straight line at a constant speed until it reaches a world boundary, at which point the UAV will "reflect" from the edge by reversing its velocity component in the direction perpendicular to the boundary, similar to how a billiard ball bounces from an edge of the table.

- *SLAF:* is an asset controller which relies on a simple UAV foliage sensor, and is based on the premise that any surveillance time spent over foliage (which inhibits target detection) is in some sense "wasted", and thus attempts to minimize UAV flight time over forested areas. The controller assumes that the only output available from the tree sensor is the percentage of foliage cover in the sensor's view-field. When the sensed canopy cover exceeds a certain threshold (currently set to 50%), the UAV reverses its course, and begins moving in the opposite direction. Should the asset be initially deployed directly over the foliage, it will continue moving in its initial direction until it leaves the foliage, or an interaction with a world boundary causes it to turn around; thus given sufficient time, the asset will find a foliage-free region. In the absence of the foliage, this controller's behavior is equivalent to the *SL* controller (*SLAF* is an abbreviation of Straight Line Avoid Forest).

- *SSLAF:* a natural extension of the SLAF strategy, the Super SLAF controller makes use of a more sophisticated foliage sensor, which can compute the spatial distribution of the trees within the sensor's view. At each time step, a SSLAF-controlled UAV uses its forest sensor to compute the percentage of foliage visible through the sensor, and if that percentage exceeds a specified threshold (the default is 10%), the asset then estimates the mass distribution of the tree canopy underneath, and moves directly away from the center of the foliage mass. Once the detected foliage percentage drops below the threshold, the asset will switch into its normal surveillance mode, continuing to move with its present velocity until an encounter with another patch of foliage or a bounce from the surveillance perimeter occurs.

- *Side:* controller produces linear "sweeping bands" of target sensor coverage by moving the UAVs back and forth along a linear path between two points on world edges. The location and direction of these surveillance corridors is determined by the initial location and velocity of each UAV, and may not include areas near the world boundary, as the assets may reverse their course right after detecting the edge of the surveillance area, and before scanning the boundary region with their target sensor.

- *Stationary:* assets hover over a fixed location in the surveillance perimeter, scanning any targets that may pass underneath. This is a simple controller that ignores the asset speed parameter in the `asset.txt` configuration file, and does not attempt to navigate around areas hidden by the foliage. Please keep in mind that the *Avoid* target strategy is 100% successful in avoiding detection by the *Stationary* assets.

As mentioned in the preceding description of the asset controllers, the maximum UAV speed is given by the `assetSpeed=`*float* parameter in the `conf/asset.txt` file, and it specifies the maximum distance (in terms of the world size units) that any given asset may travel in one simulation time step. The actual speed of an asset during the simulation might vary due to the UAV formation requirements or other constraints imposed by the specific controller. Also, because of the extensive use of the random data by some of the controllers, any particular sequence of random numbers and the corresponding change in behavior may be regenerated by using the same, non-negative value for the `assetSeed=`*integer* parameter; otherwise, a negative value for this parameter, which is the default, will cause the simulator to use the current system time in seconds, resulting in a new random asset initialization each time the simulator is run.

4 Genetic Algorithm Module

The SURVE toolkit includes a Genetic Algorithm (GA) module, whose purpose is to aid in the investigation of the impact of different force law options and parameter ranges on target detection rate. For the UAV asset controllers based on a generalized interaction potential, such as Artificial Physics (AP) or Lennard-Jones (LJ), it is sometimes possible to optimize the force function parameters in order to achieve improved surveillance performance. We accomplish this task by using genetic algorithms. Genetic algorithms can be thought of as an optimization technique, inspired by the process of the biological evolution. During execution, individual parameter sets in the population are mutated and recombined based on their performance (or fitness) in a given surveillance simulation run. The individuals with higher fitness than the average fitness of the population will survive through reproduction, and will contribute their genetic makeup to the future generations. One of the major reasons to employ a population-based stochastic algorithm is the rapid emergence of well-performing individuals, as well as the robustness of the individual performance improvements across a wide range of environmental configurations.

Typically, we use a population size of 100 individuals, and set the evolution termination criteria of the algorithm to 150 generations. The GA module generates an initial population of different potential function parameter sets using the random number seed specified in the `conf/gaapconf.txt` or `conf/galjconf.txt` files, so by changing this random seed, the program can generate many different sets of candidate parameter sets. Hence, every individual in the population represents one possible instantiation of either AP or LJ potential function optimization.

The following is a step-by-step guide on how to run the GA module, and how to set parameter settings of the potential functions using the configuration files, along with an explanation of the GA fitness function, data generation and storage, and the mechanism for visualizing the GA-evolved asset behavior.

4.1 Running the GA

The GA module included with the toolkit is located in the `ga` directory. In a UNIX-like operating environment, a user may simply execute the `runGA.sh` shell script. Otherwise, manually invoke the `java ga.PICKGA` class. Both of these options will begin evolution of parameters for the physics-based asset controller specified in the `conf/general.txt` file. If the `gavisual=`*true*|*false* parameter in the `conf/galjconf.txt` or `conf/gaapconf.txt` is set to *true*, then a window similar to the one shown in Fig. 1 (but without the information and control panels) will appear, showing asset behavior for the current

	Ga.	$F_{max}a$.	Pa.	Gw.	$F_{max}w$.	Pw.	CoeffA	CoeffB	CoeffC	Fric
Min.	100.0	1.0	0.1	100.0	1.0	0.1	5.0	5.0	5.0	0.0
Max.	5000.0	5.0	2.0	5000.0	5.0	2.0	22.0	22.0	22.0	1.0

Table 1: Ranges of Values for AP Individuals

	ϵ	$F_{max}a$.	c	d	Gw.	$F_{max}w$.	Pw.	CoeffA	CoeffB	CoeffC	Fric
Min.	1.0	1.0	1.0	1.0	100.0	1.0	0.1	5.0	5.0	5.0	0.0
Max.	50.0	5.0	20.0	20.0	5000.0	5.0	2.0	22.0	22.0	22.0	1.0

Table 2: Ranges of Values for LJ and LJMass Individual

individual parameter set. Even if the GA visualization window is initially enabled, closing it at a later time will not affect the GA run. If needed, pressing the `Ctrl+C` key combination (or the equivalent function of an Integrated Development Environment) should terminate the GA process.

4.2 Parameter Settings

The selection of the potential function for evolutionary optimization is made via the `assetControl` parameter in the `conf/general.txt` file, where the three valid evolutionary asset types are: `assetcontrol.AP`, `assetcontrol.LJ`, and `assetcontrol.LJMass`. The LJMass controller, unlike the simpler LJ algorithm, attempts to minimize the amount of time the assets spend over the foliage-covered areas.

In addition to specifying the asset controller, do not forget to check the setting of the target controller, specified by the `targetControl` parameter. Since the GA will measure the reproductive fitness of individuals based on their target-detecting rate, the choice of a target movement strategy will have a strong influence on the type of the individuals discovered through evolution.

4.2.1 Force Functions' Parameter Ranges

There are two GA-specific configuration files in the `conf` folder. Evolution of the AP assets is controlled by the `conf/gaapconf.txt` file, while both the LJ and LJMass asset behaviors evolve according to the parameters in the `conf/galjconf.txt` file. Table 1 shows the range of valid values for AP populations, and Table 2 lists the valid ranges of the LJ and LJMass individuals.

4.2.2 Explanation of the Parameters

The AP individuals:

- Ga. - gravitational force of asset-asset interactions.
- $F_{max}a$. - maximum force of asset-asset interactions.
- Pa. - exponent of the force law for asset-asset interactions.
- Gw. - gravitational force of wall-asset interactions.
- $F_{max}w$. - maximum force of wall-asset interactions.
- Pw. - exponent of the force law for wall-asset interactions.
- CoeffA - coefficient a of the second degree polynomial that determines the radius of an asset.

- CoeffB - coefficient b of the second degree polynomial that determines the radius of an asset.
- CoeffC - coefficient c of the second degree polynomial that determines the radius of an asset.
- Fric - friction in the system.

The LJ and LJMass individuals:

- ϵ - depth of the potential well.
- $\mathrm{F}_{\max}a.$ - maximum force of asset-asset interactions.
- c - non-negative attractive component.
- d - non-negative repulsive component.
- G$w.$ - gravitational force of wall-asset interactions.
- $\mathrm{F}_{\max}w.$ - maximum force of wall-asset interactions.
- P$w.$ - exponent of the force law for wall-asset interactions.
- CoeffA - coefficient a of the second degree polynomial that determines the radius of an asset.
- CoeffB - coefficient b of the second degree polynomial that determines the radius of an asset.
- CoeffC - coefficient c of the second degree polynomial that determines the radius of an asset.
- Fric - friction in the system.

4.2.3 Common Evolution Parameters

The other common parameters stored in both `gaapconf.txt` and `galjconf.txt` files are the population size (`populationsize=`*integer*), number of generations (`generations=`*integer*), number of parameters in each individual, i.e., the length of an individual (`length=`*integer*), mutation rate (`mutationrate=`*float*), crossover rate (`crossoverrate=`*float*), GA random seed (`gaseed=`*integer*), and evolution visualization preference (`gavisual=`*true*|*false*). All of these settings can be altered to fit a variety of experimental needs. Keep in mind that the negative value for the `gaseed` parameter will automatically set the pseudo-random number generator seed to the current system time, but otherwise will keep reusing the same random number sequence, resulting in the identical initial population for each new GA run.

4.3 Computing Fitness of Force Function Parameter Sets

In order to evolve the most fit individuals in the population, the simulator executes a surveillance scenario with a randomly generated forest configuration, and then calculates the fitness for each individual force law parameter set in the population. The fitness, or the performance, of an individual is the percentage of successful target localizations by the entire agent collective during the surveillance activity, whose duration is set by the `timeSteps` parameter in the `conf/general.txt` configuration file. The following formula is used when computing the fitness of an individual:

$$\text{fitness} = \frac{\text{targets seen}}{\text{total number of visible targets}} * 100$$

```
Asset Controller Requested--> AP
...Initiate GA for Physicomimetics...
Log Level 0: Basic logging information
...Growing Forest
...Forest Growth Complete: 0.0
...% found 13 % inside 100.0
...Growing Forest
...Forest Growth Complete: 0.0
...% found 21 % inside 100.0
...Growing Forest
...Forest Growth Complete: 0.0
...% found 17 % inside 100.0
...Growing Forest
...Forest Growth Complete: 0.0
...% found 12 % inside 100.0
...Growing Forest
...Forest Growth Complete: 0.0
...% found 15 % inside 100.0
GA Individual Fitness: 15.6
```

Figure 7: Sample runtime output of the SURVE GA module.

In addition, the fitness of each individual is also averaged over a specified number of different random forest configurations (currently fixed at 5), where the initial asset and target locations are different for each new forest instantiation. A sample of the runtime GA module output is shown in Fig. 7.

The selection of the individuals for reproduction is performed using the Baker's Stochastic Universal Sampling algorithm, and the offspring is generated using a one-point crossover, with the rate specified by the `crossoverrate` configuration parameter, with some additional individual changes possibly introduced via a Gaussian mutation, governed by the `mutationrate` parameter. The Gaussian random mutation used by the SURVE toolkit comes from a standard normal distribution with a 0 mean and a variance of 1. The implementation of the Gaussian mutation conforms closely to the widely accepted methodology in the evolutionary algorithms literature. In this case, the Gaussian random mutation method has a 1/(individual length) probability of mutation. Note that this implementation will reject any offspring whose parameter values fall outside of the accepted range (see Tables 1 and 2).

4.4 Data Generation and Storage

During execution, the GA module generates three different text files which contain performance-related information about the population of the force function parameters. These files are stored in the `log` folder, inside the `surveillance2` directory. Table 3 lists the names along with a brief description of the three output files generated by the evolution module.

Contents	*AP*	*LJ*	*LJMass*
All Individuals	`GAAPresults.txt`	`GALJresults.txt`	`GALJMresults.txt`
Best and Worst Individuals	`APbest_worst.txt`	`LJbest_worst.txt`	`LJMbest_worst.txt`
All Individuals with Random Seeds	`APhistory.txt`	`LJhistory.txt`	`LJMhistory.txt`

Table 3: Output files from the GA module.

Each output file contains the sequential generation number, as well as the sequential identification of each individual in the population along with that individual's fitness (i.e., surveillance performance). The `best_worst` file only shows the best and the worst performing individuals for each generation, and is helpful in evaluating the efficacy of the evolutionary strategy. The history file contains the generation count, an individual's location in the population, and an individual's random seed, but does not show the fitness of the individuals. Each output file lists all of the force function parameter values represented by each GA individual.

4.5 Using GA Individuals

The SURVE toolkit provides the functionality to directly process the output files generated by the GA module by first loading the entire history file, and then instantiating the asset controller with the parameters from the evolved individual. Once the GA completes its run and generates the output files, the GA menu can be used to process that output and to conduct experiments with the evolved individuals (for more information on the GA menu, please see Section 6 on simulator menus). The GA menu contains two actions: Load GA History and Load GA Individual. To create an asset controller based on a GA individual, first load the GA history file, and then load a specific GA individual by specifying both the generation and individual identification tags. The Load GA History menu item will display a "File Open" dialog window initially positioned in the `log` directory, through which the user should select the correct GA history file and then press the Open button to load the history file into the SURVE application. Then choose the Load GA Individual menu item, which will display a window similar to the one shown in Fig. 8, where the generation and individual identification numbers can be specified. Once this information is entered, the user can observe this individual's surveillance behavior by simply clicking the Start button on the main control panel of the simulator (see Fig. 1).

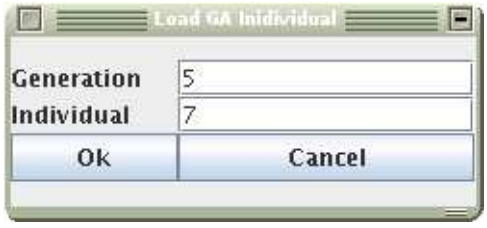


Figure 8: Load GA Individual window.

5 Simulator Parameters

Upon startup, SURVE reads several text-based configuration files from the `conf` directory to initialize its components. The list below contains the names of all of the files, a brief description of the simulator functionality that depends on that file, and then provides a detailed explanation of each configuration parameter, its default value, and in parenthesis, the type of the parameter. One important fact to remember is that configuration parameter values are only examined during SURVE initialization, and changing the parameter values in the configuration files while the simulator is running will have no effect on the currently running instance of the simulator. To change system parameters at run time, please use the control panel sliders and option menus in the SURVE graphical user interface (see Section 6).

5.1 General Parameters

The `conf/general.txt` configuration file provides values for target control parameters, as well as the general SURVE simulator parameters.

- `agents=5` (*positive integer*) - number of UAV assets in the simulation.
- `targets=100` (*non-negative integer*) - number of targets in the simulation.
- `targetSeed=-1` (*integer*) - target random number generator seed; system time is used when the value is negative.
- `targetSpeed=0.4` (*non-negative float*) - maximum speed of the targets.
- `targetOffset=-1.0` (*float*) - target initial position horizontal offset in terms of the world size.
- `targetControl=targetcontrol.Sneaky` (*string*) - target movement strategy, entered as a fully-qualified Java class name.
- `assetControl=assetcontrol.SL` (*string*) - asset control strategy, specified as a fully-qualified Java class name.
- `size=200` (*positive integer*) - world size (in screen pixels).
- `timesteps=1000` (*positive integer*) - number of simulation steps in each surveillance scenario.
- `sleeptime=0` (*non-negative integer*) - number of milliseconds between screen updates in visual experiments.
- `debug=0,3` (*integer list*) - selection of debugging events to be reported to the console. Currently, four types of debugging events are available: (0) basic logging information, (1) random seed information, (2) step-by-step target localization information, (3) target collision errors.

Please keep in mind that both asset and target controller strategies must be named in the configuration file with their fully-qualified Java class names (i.e., the name of the strategy must be prepended with the `assetcontrol` or `targetcontrol` Java package name).

5.2 Asset Parameters

The controlling parameters for the UAV surveillance assets are stored in the `conf/asset.txt` file, and are enumerated below:

- `assetSpeed=3.0` (*positive float*) - maximum speed of an asset.
- `targetRadius=5` (*positive integer*) - radius of the ground-level target sensor footprint.
- `targetSensorType=circle` (*string*) - shape of the ground-level target sensor footprint; possible values are *circle* and *square*.
- `targetSensorIncludeBounds=0` (*integers 0 or 1*) - Boolean flag which controls the distance comparison operator used during target detection; when the parameter value is 0, operator $<$ is used, otherwise $\leq$ comparison is employed.
- `foliageRadius=20` (*positive integer*) - radius of the ground-level foliage sensor footprint (always treated as a circle).

- `detectionProb=1.0` (*float between 0 and 1*) - probability of successfully detecting a target when it is within the sensor range.
- `assetSeed=-1` (*integer*) - random number generator seed for stochastic components of asset behavior; if the value is negative, then the current system time in seconds will be used.

Initial asset positions are selected randomly from either a uniform or a normal distribution, depending on the asset controller. Most of the time, an asset will move with the speed given by the value of the `assetSpeed` parameter, unless interactions with the neighboring assets cause the UAV to slow down.

5.3 Forest Parameters

The foliage production procedure uses the values of the following parameters, which are located in the `conf/forest.txt` configuration file:

- `growthStrategy=env.NaturalPineCones` (*string*) - fully-qualified Java class name for the tree placement strategy; currently, the only valid choice is `env.NaturalPineCones`.
- `seedRadius=8` (*positive integer*) - maximum distance that a seed can fall away from its parent tree.
- `seedStrength=0.2` (*float between 0 and 1*) - probability that a seed will grow into a tree (i.e., seed viability).
- `seedDecay=0.9` (*positive float*) - seed viability decay rate.
- `forestSize=10` (*positive integer*) - initial size of the forest (i.e., the number of starter trees, randomly distributed over the surveillance area).
- `forestCoverage=0.2` (*float between 0 and 1*) - desired amount of foliage, expressed as the fraction of the total area of the simulated world.
- `forestSeed=-1` (*integer*) - random number generator seed for the forest algorithm; system time will be used as the seed when this value is negative.

The first step of the forest generation process selects `forestSize` initial locations randomly, in order to plant the first parent trees. During the next step, the parent trees will produce seeds within the circle of `seedRadius`, and each seed will have a `seedStrength` probability of growing into a new tree. Note that if multiple nearby parent trees deposit seeds in the same location, then the probability that a new tree will appear at that location increases in direct proportion to the number of seeds planted there. Since not all of the planted seeds will become trees during the first iteration, the seeds that failed to produce a new tree will lose some of their vitality, with the rate of seed viability decline given by the value of the `seedDecay` parameter. The seeding algorithm continues until at least `forestCoverage` of the entire surveillance area is occupied by trees, thus it may be possible for the final foliage cover to exceed the specified threshold by a small amount.

6 User Interface

This section provides an overview of the SURVE graphical user interface (GUI). Whenever possible, the design strives for simplicity and intuitive user experience, so many of the options and buttons should be self-explanatory. Because of the desire to keep the interface simple, only a subset of the simulator parameters can be changed via the GUI; the rest of the configuration should be performed through the configuration file interface (see Section 5 for more information).

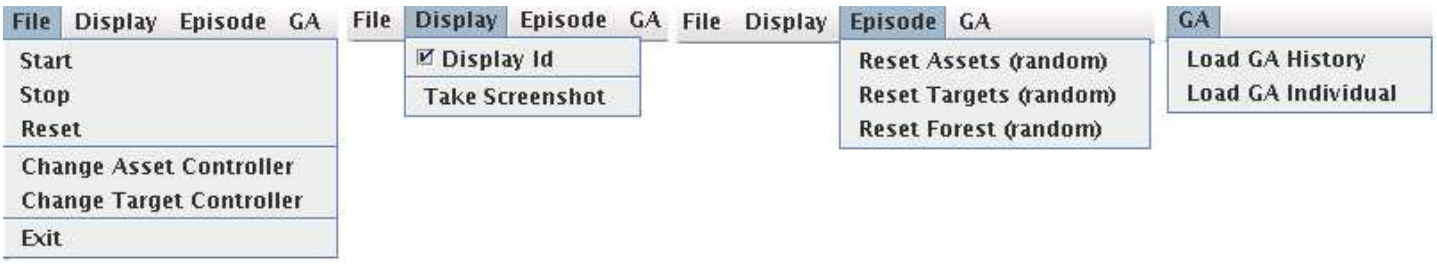


Figure 9: Expanded view of the SURVE application menubar.

6.1 Menubar

The main application menubar provides the means to quickly and easily change the most important simulation parameters, and to perform large-scale changes to the surveillance experiments, such as altering the asset control strategy, or building a new foliage map. A composite image of the expanded view of the main SURVE menus is shown in Fig. 9. The simulator can also be operated via the the button and slider control panel, shown in Fig. 13.

6.1.1 File Menu

The File menu contains frequently used simulation commands: start and stop will initiate or terminate the simulation, the reset will terminate the current simulation and reposition the assets and the targets using the current system time as the random number generator seed. The same functionality is also available through the three buttons in the control panel (see Fig. 13). The menu items for changing asset and target controller strategies will bring up a new window where the new controller type can be selected (see Fig. 10). All of the default asset and target controllers that are distributed with the SURVE simulator can be activated by using these menus. Selection of a new controller will cause the corresponding random number generator to be initialized with the current system time, and the position of the agents will be reset. As the name suggests, the Exit menu item will terminate the simulation and quit the SURVE application.

6.1.2 Display Menu

The Display menu allows the user to toggle the on-screen marking of each asset with its identification number, which is extremely helpful when tracking the trajectory of a specific asset. The Take Screenshot item creates a JPEG image of the current SURVE state, pausing the simulation if necessary. Only the

Figure 10: Selecting a new asset and target controller.

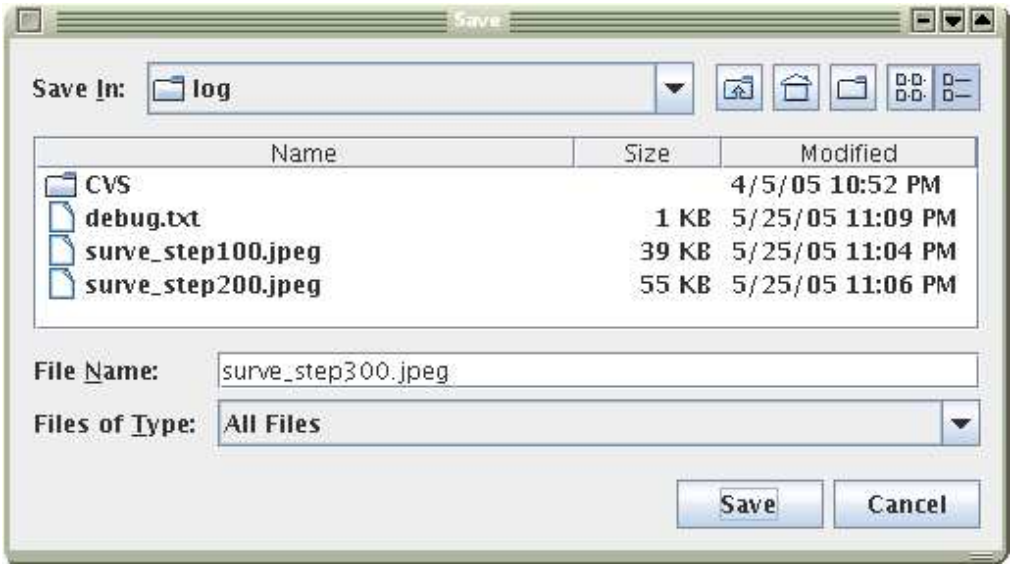


Figure 11: Saving a screenshot of the SURVE simulation.

surveillance area, including the forest, assets, and targets is saved in the image; the GUI elements and simulator controls are not captured in the image (to obtain a complete screenshot of the entire SURVE window use a capture utility provided by the operating system). Clicking the Take Screenshot button will open a "Save File" dialog box (see Fig. 11), where the file name for the new image should be specified.

6.1.3 Episode Menu

The Episode menu provides a quick way to regenerate a forest, reposition targets, and to re-initialize the UAV assets. The first menu action Reset Assets will set the new asset positions using the current system time as the seed for the random number generator. Executing this action will relocate the assets, but will leave the targets in their current positions. However, the count of targets detected by the assets will be set to zero, since resetting the assets creates a completely new set of surveillance agents. Use the Reset Targets menu item for restarting the targets, which again will use the current system time as the random seed, and will place the new targets using this new random number sequence. The old UAV asset locations will not be affected, but since a completely new target group is created as the result of executing this action, the count of targets previously detected will again be set to zero. The last function in this menu, Reset Forest, will change the forest map by obtaining the new random number generator seed from the current system time, and will regrow the forest with the same area coverage as before, but now with different locations of the starter trees. While both the assets and the targets will keep their current positions after the change in the foliage map, all of the counters (including detected targets and simulation time) will be reset to zero.

6.1.4 GA Menu

The GA menu can only be used when the GA module execution history output file is available (see Section 4 for a detailed explanation of the SURVE evolutionary algorithm feature). The purpose of this menu is to allow for a run-time visualization of the asset surveillance behavior based on the force function parameters obtained with the genetic algorithm. In order to visualize a given parameter set, you need to know the generation and the individual identification number of the GA individual you wish to instantiate. Once these are known, choose the Load GA History menu item, which will display the "Open File" window similar to the one shown in Fig. 12, where you should select the correct history file, and then click the Open button to load all of the parameter sets into the application. Then, activate the GA menu again, and this time

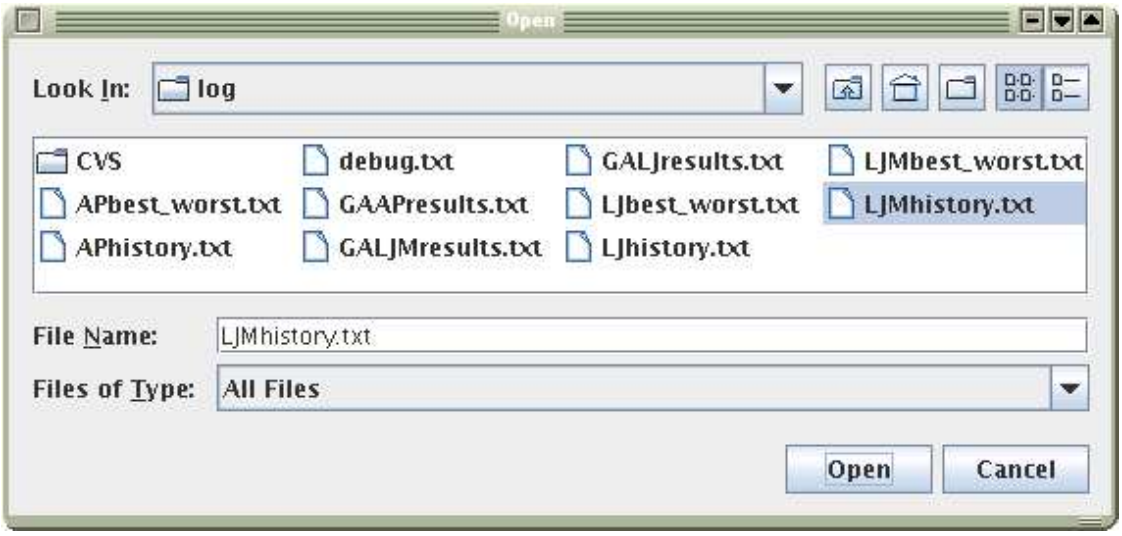


Figure 12: Loading GA evolution history file.

select the Load GA Individual option, which will display the entry dialog (see Fig. 8) where you will input the generation and individual ID tags for the specific parameter set you wish to load. When you click the OK button, the asset controller parameters will be extracted from the GA history file and activated, which will reset the asset positions and update the asset control slider panel. *Note:* before loading the GA history output file, you must make sure that the current asset controller is of the same type as the asset controller used to generate the GA history file; in other words, if you load the `LJhistory.txt` file, then make sure that the current asset controller is set to `assetcontrol.LJ`. For greater convenience, you may select the Load GA Individual option multiple times with different generation and individual ID tags without restarting the simulator, but keep in mind that the current target locations and the forest map are preserved when you load a new set of asset control parameters. If you wish to change the target or foliage placement, please use the appropriate item from the Episode menu. Also, if you mistakingly activate the Load GA Individual item before loading a history file, then SURVE will first prompt you for the history file, and then will allow you to select an individual from that file.

6.2 Simulation Control Panel

The main SURVE control panel is shown in Fig. 13. The four buttons on the left hand side of the panel are available for user convenience, and are identical in functionality to the corresponding File and Display menu items. The panel also contains an information area where the current asset controller and the target display legend are shown. Note that the target display symbol legend, located to the right of the simulation control buttons, is useful in identifying the four different surveillance classifications that each target may have, which are:

1. visible, but not detected (downward triangle symbol)
2. detected (plus sign symbol)
3. undetected target obscured by foliage (multiplication symbol)
4. detected target obscured by foliage (vertical pipe symbol)

Below the legend, you will find the information about the current simulation, which includes simulation time step, the number of detected targets, and the number of targets that have been visible at least once

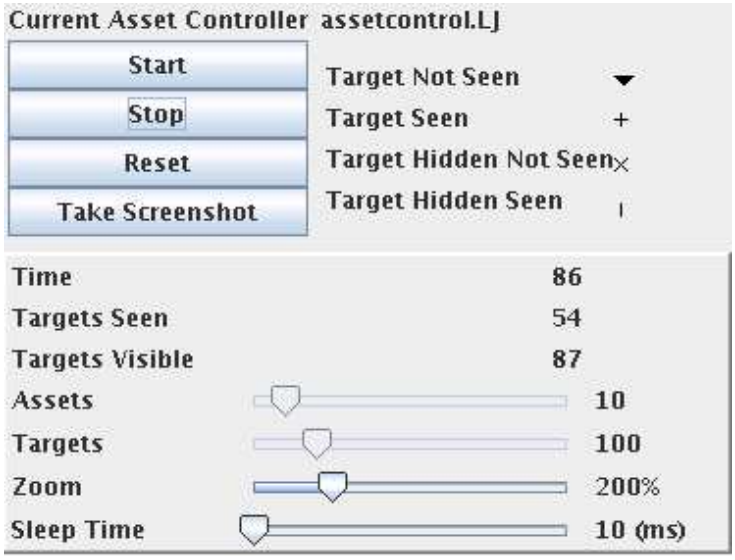


Figure 13: SURVE control panel.

up to this time step. For stationary targets, the visible counter will not change as the simulation runs, but for moving targets this number will normally increase as the simulation progresses, since more and more targets will be moving through the open surveillance areas.

Underneath the simulator statistics are two disabled sliders for the number of assets and targets. These numbers cannot be changed while the simulator is running, and should only be set in the `conf/general.txt` file as the values of the `agents` and `targets` parameters. The zoom control bar allows for magnification of the surveillance display region, however, the display region will always use as much screen space as possible, thus it may be necessary to resize the SURVE window in order to shrink the display in some cases. The control panel also includes the slider for adjusting the sleep time, which affects the delay between the consecutive screen updates, and increasing this delay will be helpful when tracing the trajectory of individual assets or targets.

6.3 Asset Controller Panel

When you replace the asset controller strategy via the File menu, you will notice that the slider control panel in the lower right of the SURVE application window changes (see Fig. 14). This happens because

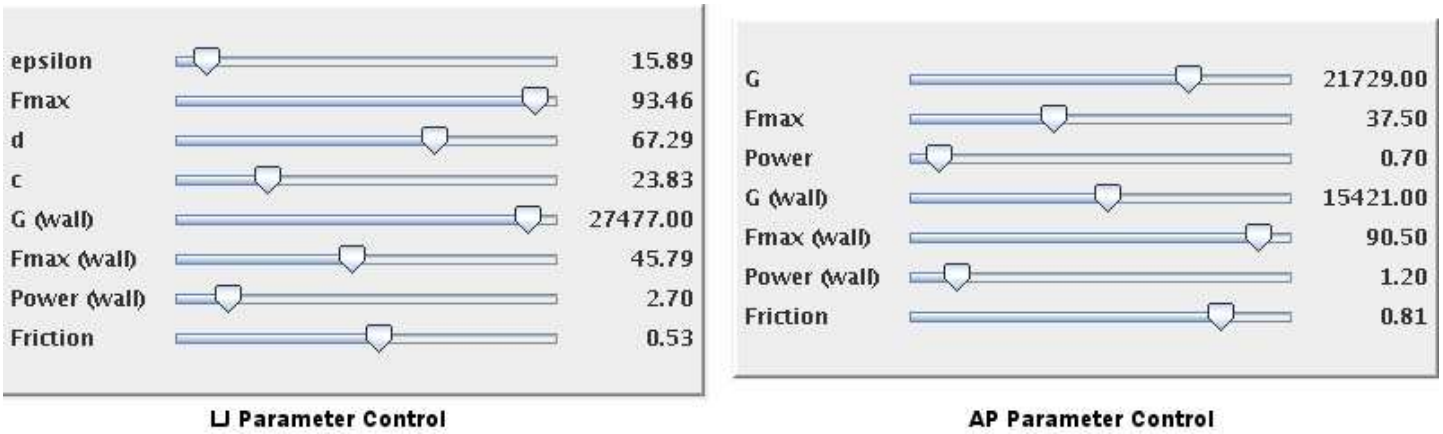


Figure 14: SURVE asset controller adjustment panels.

each controller has a different set of parameters that may be modified during the run of the simulation, and some controllers may have no parameters, resulting in an empty GUI panel.

Asset controllers based on physical interactions between neighboring assets and the environment provide the most flexible and powerful control interface, through which the user can alter the strength and type of asset-to-asset and asset-to-environment forces. The two parameter sliders available in the control panel, which do not appear in the canonical AP and LJ force equations, are the $F_{\max}$ and the friction controls. Parameter $F_{\max}$ limits the force resulting from assets moving too close to each other or the surveillance perimeter. The friction parameter determines how much of the assets' kinetic energy is dissipated during one simulation time step.

7 Batch Mode Experiments

One extremely powerful feature of the SURVE toolkit is the ability to run a large number of surveillance experiments in the non-interactive batch mode. Instead of displaying the graphic user interface and rendering every detail for each simulation time step, the batch mode simply executes the simulation with visualization disabled, and writes the performance statistics to an output file for later processing and analysis. Because this mode of surveillance experimentation requires a great deal of customization of system parameters, the default SURVE distribution includes a sample batch mode class file called `IEEETester` which illustrates how you can create new experiments to better fit your investigation.

7.1 IEEE Tester (batch mode)

The `IEEETester` class file will execute a number of SURVE simulations for a given foliage cover, number of UAV assets, and target detection probability, overriding the corresponding parameters from the configuration files. Its purpose is to evaluate the robustness of a surveillance strategy by repeatedly executing it for a combination of different foliage maps and initial target positions. The format of the text output file follows the MATLAB M-script conventions, with the surveillance simulation results stored as 3 row vectors of floating point numbers, which correspond to the percentage of visible targets found by SL, SLAF, and LJ asset controller strategies. You can change the number of foliage maps generated in each run by modifying the initialization value of the `forestRuns` variable in the `experiments/IEEETester.java` source file. Likewise, the number of different target starting positions (per forest map) is controlled via the `testRuns` variable in that source file. After making changes to the source code, remember to compile the file with `javac experiments/IEEETester.java`. If working on a UNIX-like operating system, the `Makefile` included in the SURVE source distribution can be used to quickly rebuild the application.